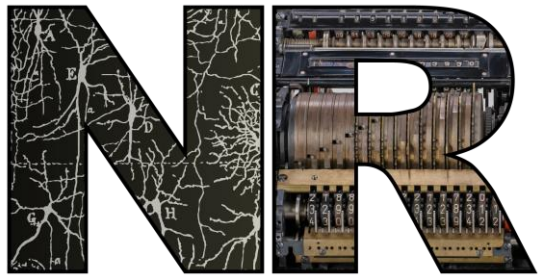
Cosyne Tutorial 2022

Spiking Neural Network Models in Neuroscience

Dan Goodman

Imperial College London

Imperial College London



neural-reckoning.org

Course outline

Part 1 – Classical spiking neural networks

- Biology / what is an SNN
- Neuron models (LIF, HH)
- Brief tour of neuron dynamics
- Coincidence detection

Exercises:

- Improve on my sound localiser
- Competition!

Part 2 – (Machine) learning with SNNs

- Learning with SNNs
- Machine learning and gradient descent
- Surrogate gradient descent for SNNs
- Coding it in PyTorch
- A surprise

Exercise:

- Implement sound localisation with surrogate gradient descent

Course “philosophy”

Practical

Aim: after this course, you should have an idea of how to start using SNNs

Not enough time to talk about everything, so I will point to further resources

Not much theory. You can find this in many good textbooks already.

Will therefore miss out a LOT.

Computation oriented

Brain uses SNNs to carry out computations.

A lot of research focusses on properties of spiking neurons. Important, but…

Aim: how do those properties contribute to useful computations?

Useful can mean for the brain, and maybe also for machine learning (*maybe!*).

So why should I care about SNNs?

Neuroscience.
SNNs are what the brain does. If we want to fully understand the brain we need to understand SNNs.

Intelligence.
They're the basis of the only known system for general intelligence.

Intellectual challenge.
Find a way to think about hybrid continuous / discrete system, and computation based on this.

The coming revolution.
Exciting times for SNNs with new ways to train them (will talk about it in second half).

Neuromorphic hardware.
Low power consumption.

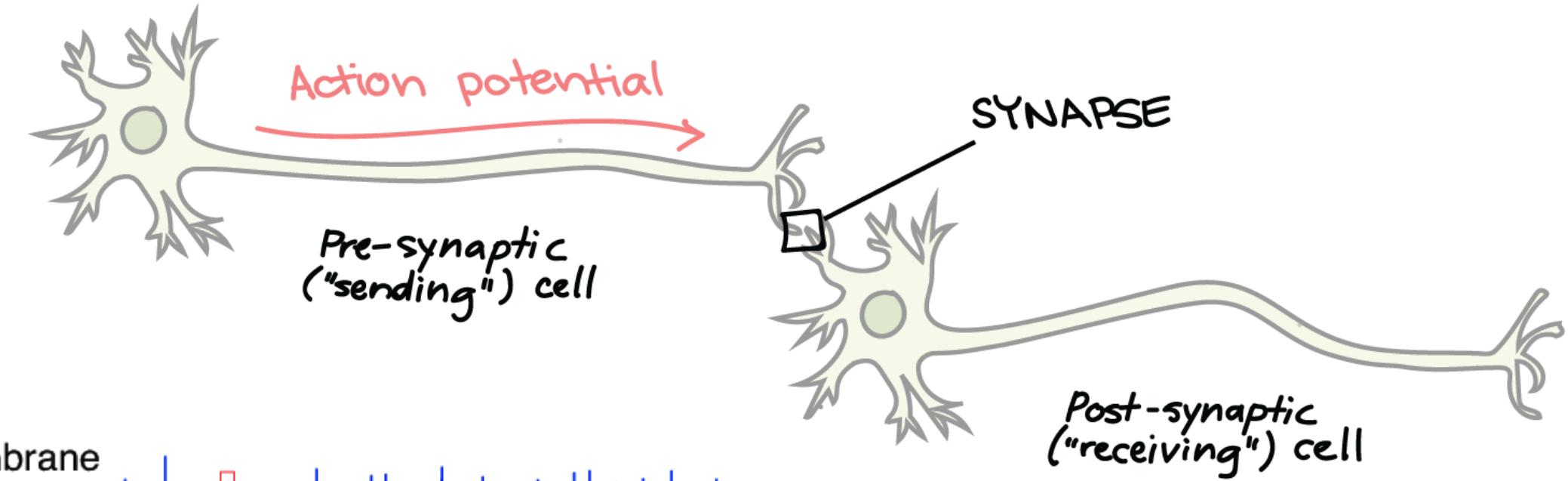
Advantages of computation with SNNs?

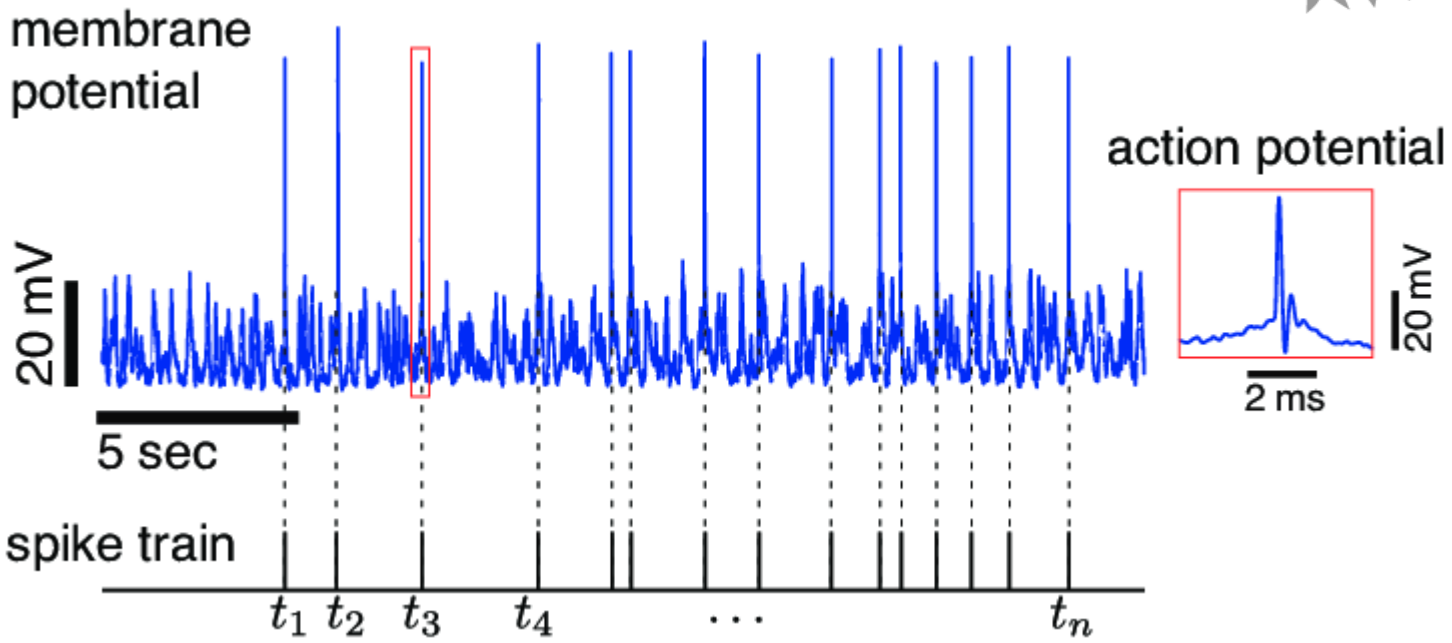
Speculative, but my guesses are…

- Fast computation / rapid decision-making
- Multiplexing

Part 1 – “Classical” spiking neural networks

What is a spiking neural network?



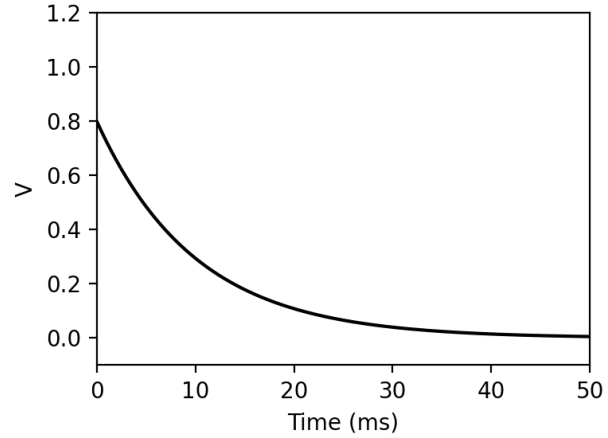


A simple model: the leaky integrate-and-fire (LIF) neuron

Membrane potential V evolves according to a differential equation

$$\tau \frac{\mathrm{d}V}{\mathrm{d}t} = -V$$

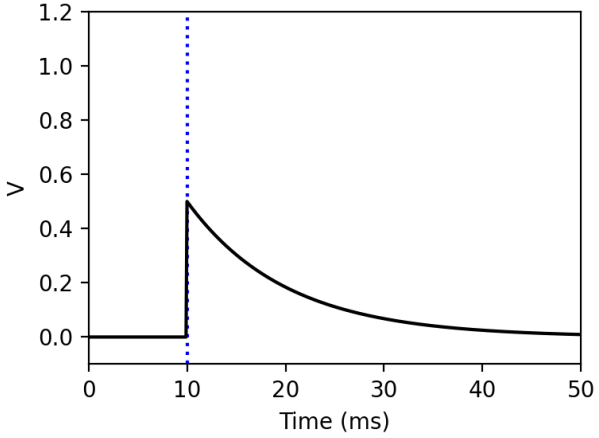
Leak



When a neuron receives a spike, V increases by synaptic weight w:

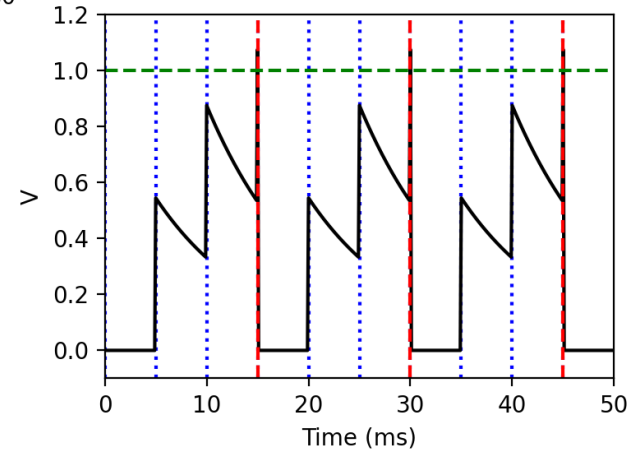
$$V \leftarrow V + w$$

Integrate



When $V > V_t$ the neuron "fires a spike" and resets:

$$V \leftarrow 0$$



Nonlinear, discontinuous dynamics!

How to simulate an LIF neuron?

Let’s take a look at some code

Slightly more complicated model: 2D LIF

Add dynamic threshold V_t

Threshold dynamics:

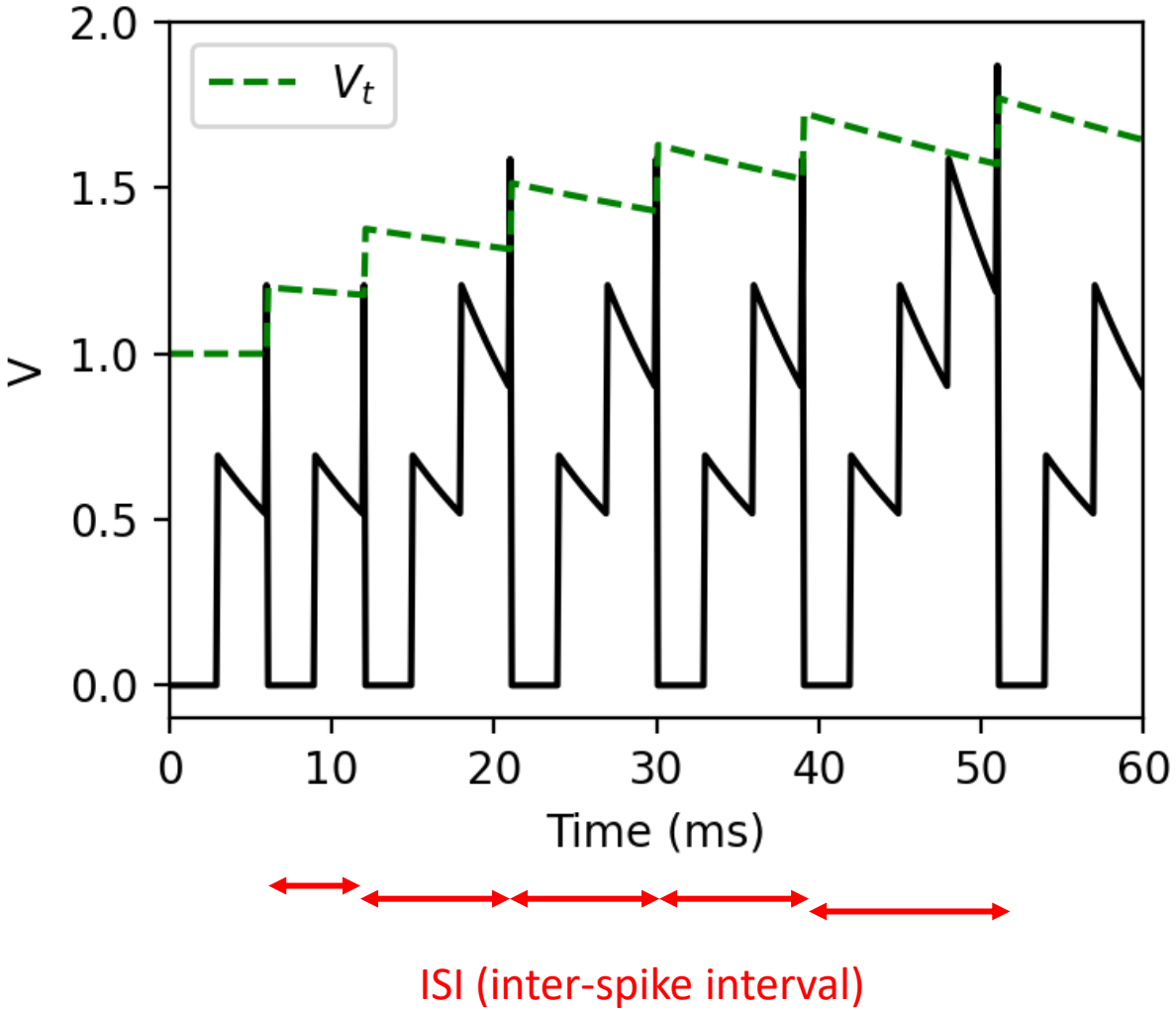
$$\tau_t \frac{\mathrm{d}V_t}{\mathrm{d}t} = 1 - V_t$$

New spike threshold condition:

$$V > V_t$$

After a spike:

$$V \leftarrow 0$$
$$V_t \leftarrow V_t + \delta V_t$$



Hodgkin-Huxley and other biophysically detailed models

Currents from many different ion channels

Each has an equation, add them together

Gives BIG equations, hard to understand

$$I = C_m \frac{\mathrm{d}V_m}{\mathrm{d}t} + \bar{g}_{\mathrm{K}} n^4 (V_m - V_K) + \bar{g}_{\mathrm{Na}} m^3 h (V_m - V_{Na}) + \bar{g}_l (V_m - V_l),$$

$$\frac{dn}{dt} = \alpha_n(V_m)(1-n) - \beta_n(V_m)n$$

$$\frac{dm}{dt} = \alpha_m(V_m)(1-m) - \beta_m(V_m)m$$

$$\frac{dh}{dt} = \alpha_h(V_m)(1-h) - \beta_h(V_m)h$$

$$\alpha_n(V_m) = \frac{0.01(10-V)}{\exp\left(\frac{10-V}{10}\right)-1} \qquad \alpha_m(V_m) = \frac{0.1(25-V)}{\exp\left(\frac{25-V}{10}\right)-1} \qquad \alpha_h(V_m) = 0.07\exp\left(-\frac{V}{20}\right)$$

$$\beta_n(V_m) = 0.125\exp\left(-\frac{V}{80}\right) \qquad \beta_m(V_m) = 4\exp\left(-\frac{V}{18}\right) \qquad \beta_h(V_m) = \frac{1}{\exp\left(\frac{30-V}{10}\right)+1}$$

I prefer to use reduced (usually 1d or 2d) LIF models

Derive reduced models via various mathematical assumptions (e.g. slow/fast)

These actually seem to fit the data better!

Whistle stop tour into the world of neuron dynamics

Bursting

Mechanisms

Many different types and mechanisms

Interaction of slow and fast dynamics

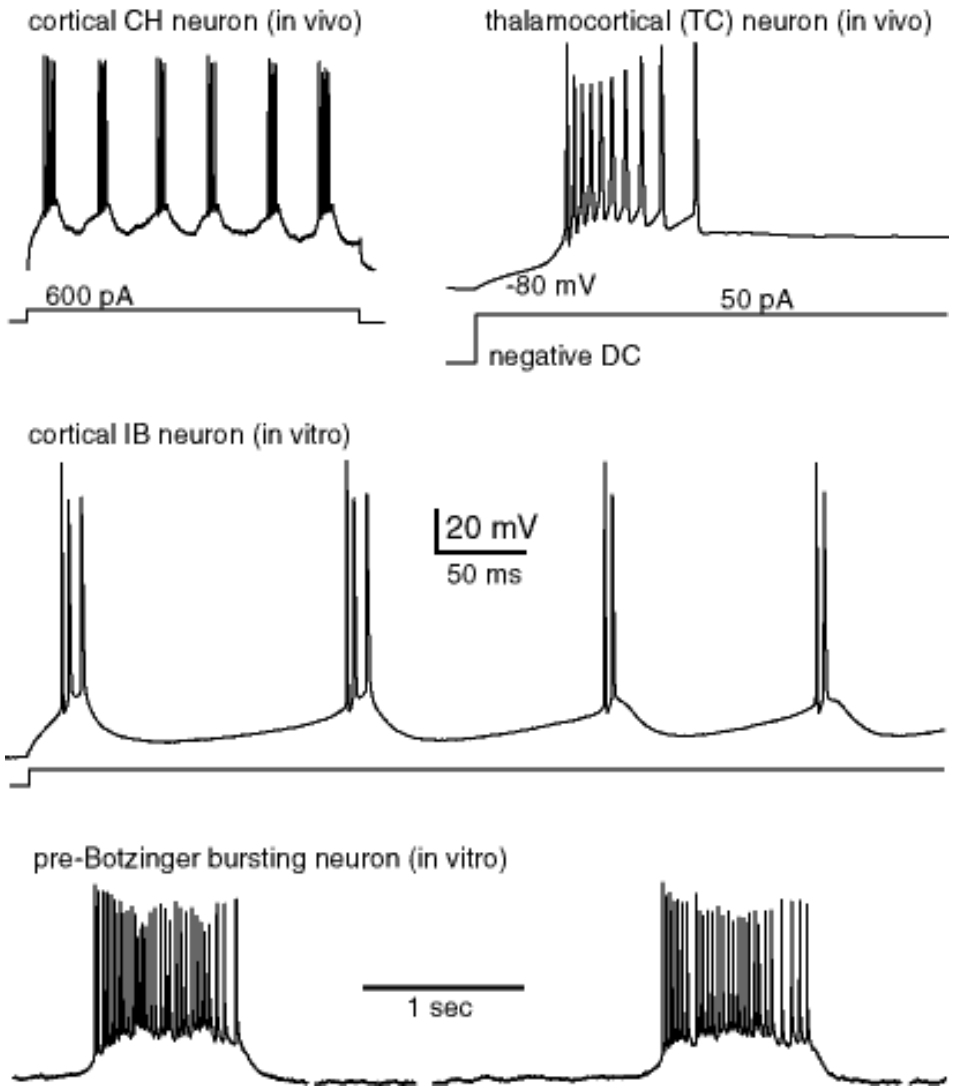
Possible roles

Reliability and signal to noise ratio

Multiplexing

Back-prop of error signal (Payeur et al. 2021)

(See also work by Rui Ponte Costa at this conf!)



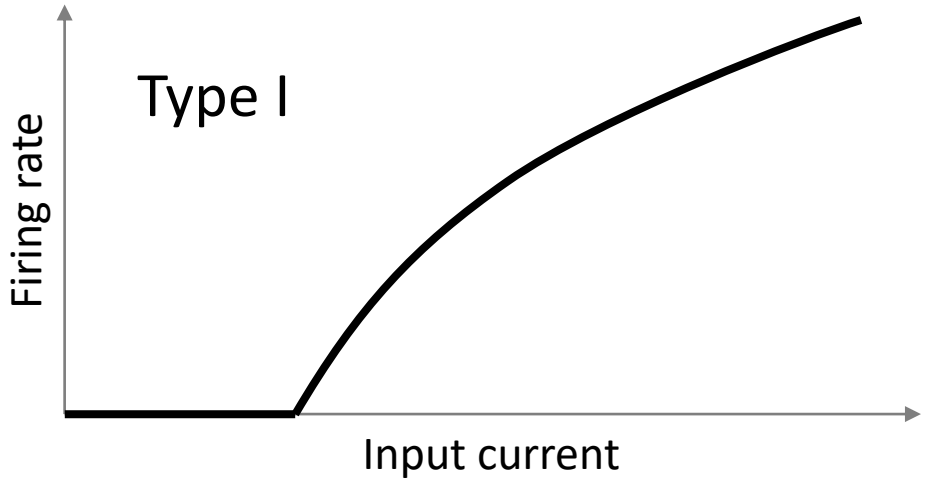
Whistle stop tour into the world of neuron dynamics

Type I/II

Mechanisms

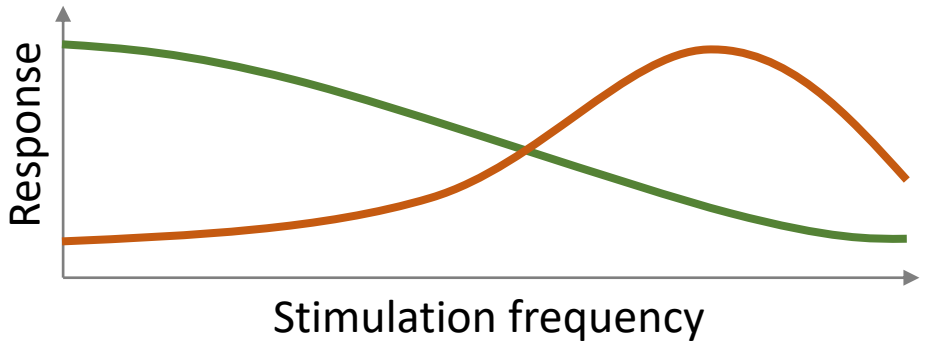
Multiple biological mechanisms

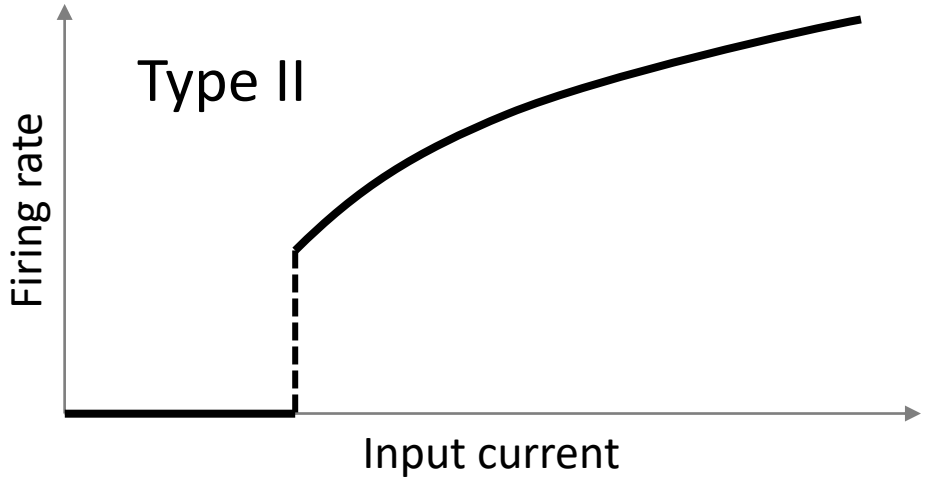
Analyse with bifurcation type



Possible roles

Integrator / resonator





Whistle stop tour into the world of neuron dynamics

Spatial structure

Mechanisms

Active/passive dendrites

Location of inputs

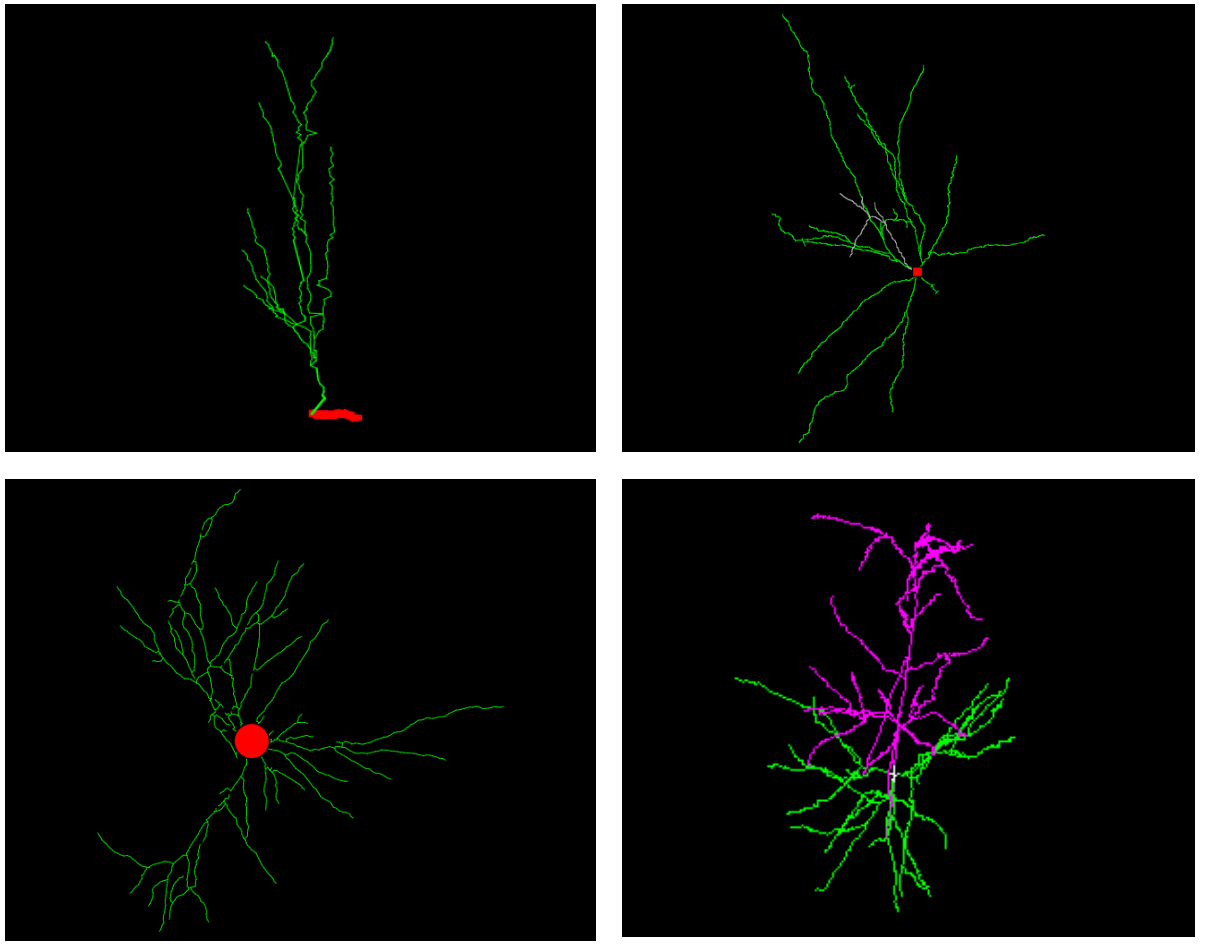
Possible roles

Too many to list

Still very active, e.g.

A single neuron can solve MNIST (Jones and Kording 2021)

Some neurons from neuromorpho.org



Coincidence detection and exercise

Let’s take a look at some code.

Part 1 - Exercise

Try to improve on my solution.

Competition: best solution presents what they did.

Divide yourself into pairs / each table can discuss.

Part 2 – (Machine) learning with SNNs

So how do spiking neural networks learn?

Short answer: we still don’t know.

Break it down into two slightly separate questions:

1. How do biological neurons learn?
2. How can we train spiking neural network models to do tasks?

Expect these two questions to shed light on each other.

Each is worth pursuing independently.

Will focus primarily on (2) in this tutorial.

Biological learning

General approaches

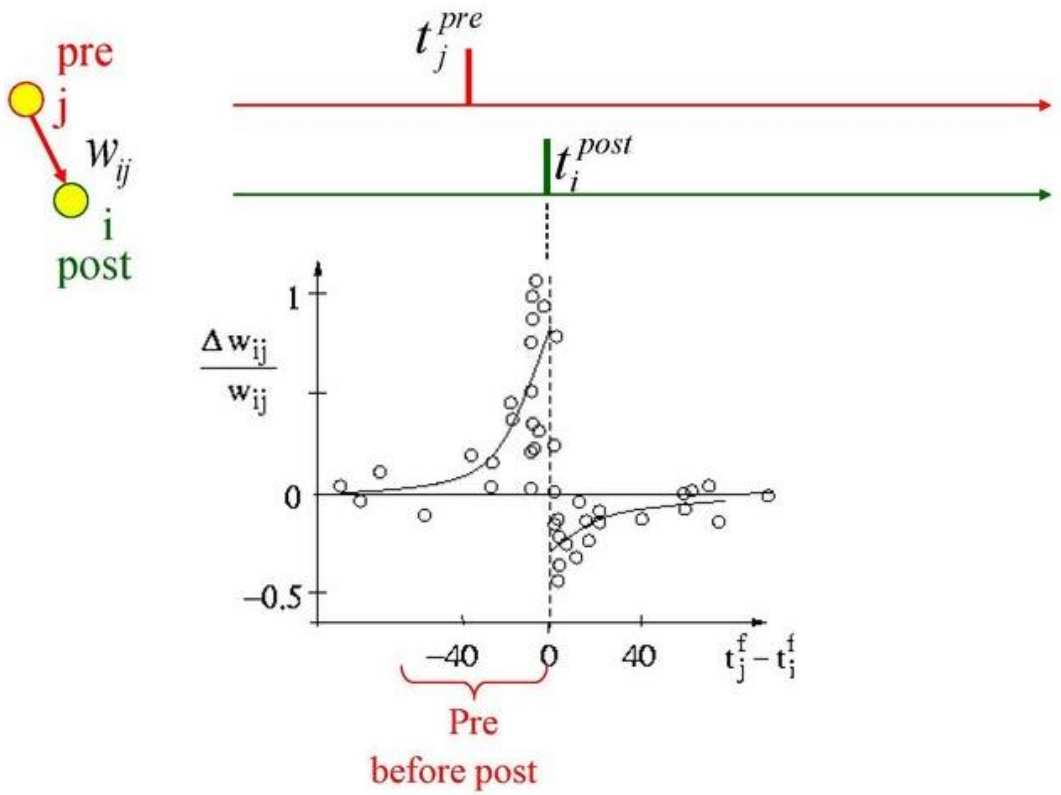
- Strengthen/weaken synapses
- Add/remove synapses
- Add/remove neurons
- Other wild stuff (packets of RNA)?

Some models

- STDP
- Reward-modulated

None of these work very well for hard tasks.

(So are they good models of the brain?)



Solving hard tasks with SNNs

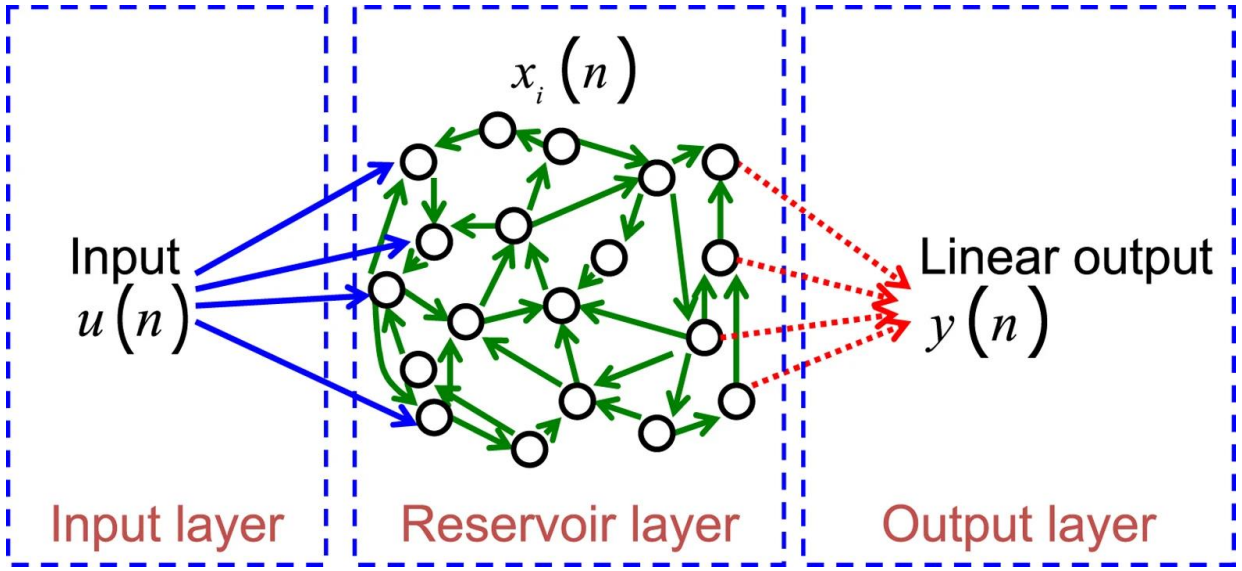
Some ideas that are still active (there are many others)

Reservoir computing (Maass, …)

Also known as:

- Echo state networks
- Liquid state machines

Performance limited in practice?



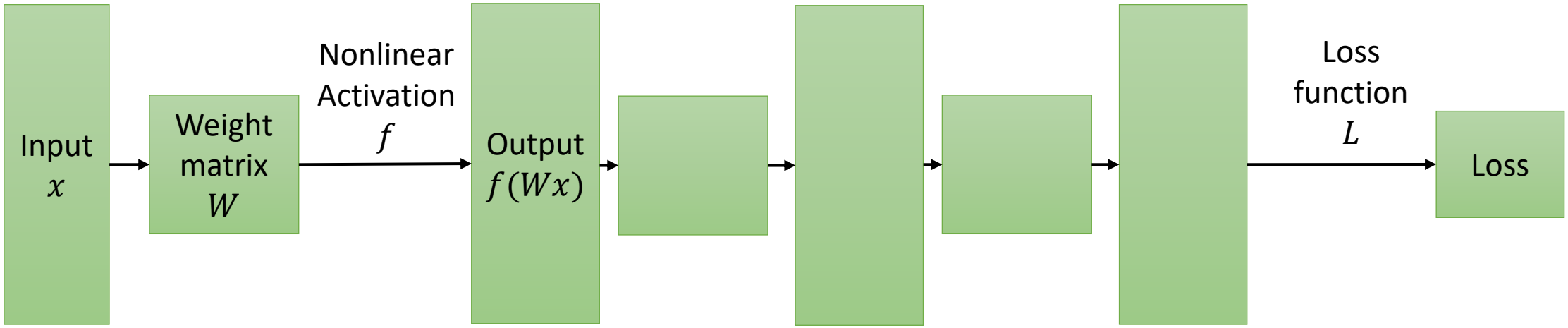
Evolutionary optimisation (Schuman, …)

Computationally expensive

Surrogate gradient descent (Zenke, …)

Will come back to this later

Very brief interlude on artificial neural networks

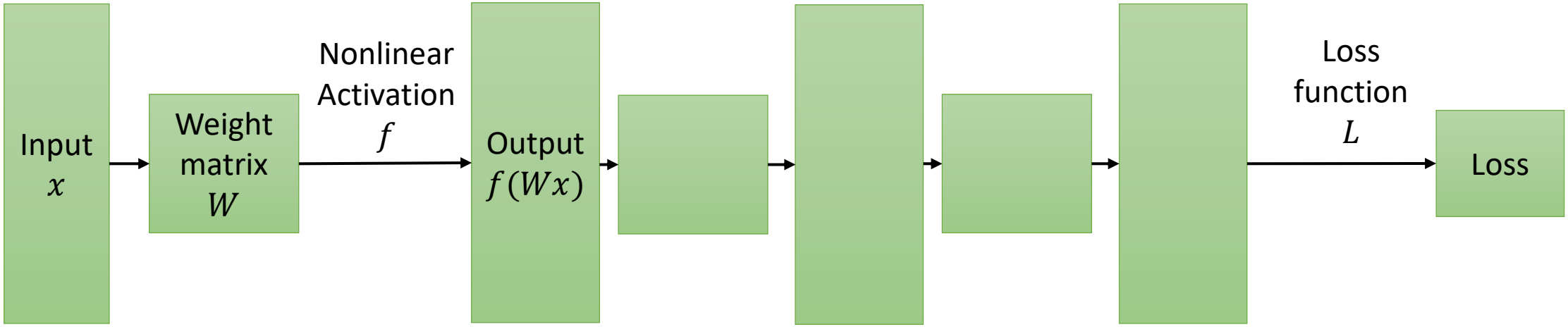


Loss is a big function $L(\theta, x)$ where θ is vector of all learnable parameters
Composed of lots of simple functions (matrix multiply, sum, …)

Compute gradient of loss with respect to parameters: $(\nabla L)_i = \frac{\partial L}{\partial \theta_i}$

Gradient descent with learning rate α: $\theta \to \theta - \alpha \nabla L$

Very brief interlude on artificial neural networks



Chain rule

$x \xrightarrow{f} y \xrightarrow{g} z$ or
$z = g(f(x))$ gives

$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$ or
$z' = g'(f(x)) \cdot f'(x)$

Chaining the chain rule

$x_1 \to x_2 \to \cdots \to x_n$

Gives

$\frac{dx_n}{dx_1} = \frac{dx_n}{dx_{n-1}} \cdot \frac{dx_{n-1}}{dx_{n-2}} \cdots \frac{dx_2}{dx_1}$

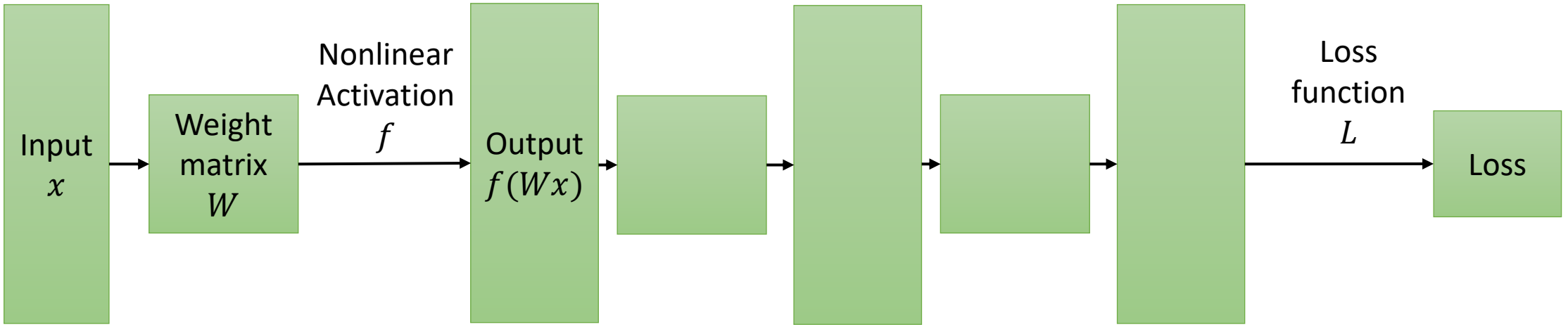
Vector version

$x \in \mathbb{R}^n \xrightarrow{f} y \in \mathbb{R}^m \xrightarrow{g} z \in \mathbb{R}^p$

Jacobian matrix $J_{ij}^f = \frac{\partial f_i}{\partial x_j}$

Chain rule is $J^{g \circ f} = J^g J^f$

Very brief interlude on artificial neural networks



Loss is a big function $L(\theta, x)$ where θ is vector of all learnable parameters
Composed of lots of simple functions (matrix multiply, sum, …)
Compute gradient of loss with respect to parameters: $(\nabla L)_i = \frac{\partial L}{\partial \theta_i}$
Gradient descent with learning rate α: $\theta \to \theta - \alpha \nabla L$

Chain rule

$x \xrightarrow{f} y \xrightarrow{g} z$ or
$z = g(f(x))$ gives

$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$ or
$z' = g'(f(x)) \cdot f'(x)$

Chaining the chain rule

$x_1 \to x_2 \to \cdots \to x_n$

Gives

$\frac{dx_n}{dx_1} = \frac{dx_n}{dx_{n-1}} \cdot \frac{dx_{n-1}}{dx_{n-2}} \cdots \frac{dx_2}{dx_1}$

Vector version

$x \in \mathbb{R}^n \xrightarrow{f} y \in \mathbb{R}^m \xrightarrow{g} z \in \mathbb{R}^p$

Jacobian matrix $J^f_{ij} = \frac{\partial f_i}{\partial x_j}$

Chain rule is $J^{g \circ f} = J^g J^f$

☜ Forget all this

Autograd.

Write the forward pass, and PyTorch will calculate everything for you.

+ better algorithms than gradient descent

Surrogate gradient descent

So let's try this for SNNs.

Problem

Threshold is $v > 1$ or Heaviside $H(v - 1)$

$H'(v) = 0$ for all $v \neq 0$

Gradients are all zero.

Solution 1/n: smooth threshold

Huh and Sejnowski (2018)

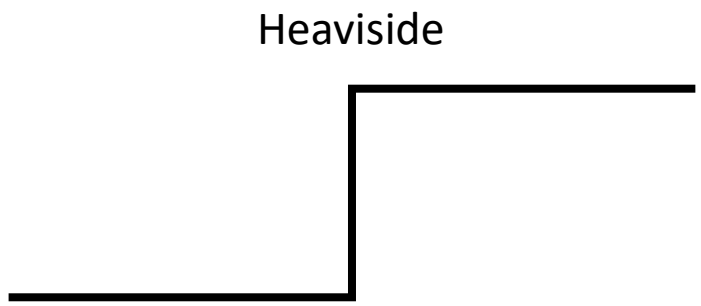
Works, but slow and not an SNN

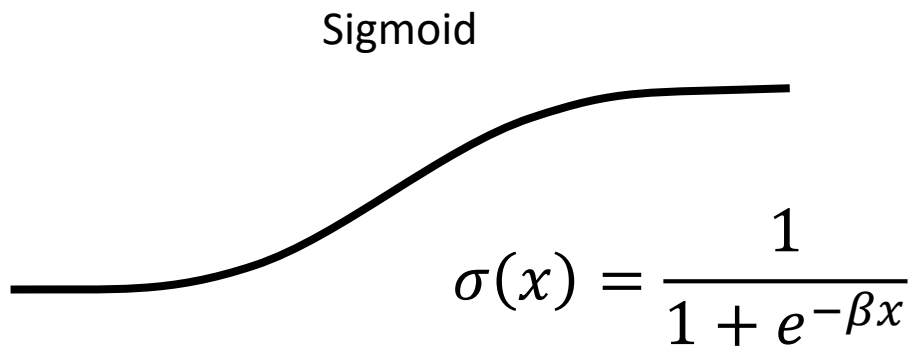
Solution 2/n: surrogate gradient descent

Neftci et al. (2019)

Only smooth gradient (backwards pass)

This is crazy! It shouldn't work, but it does. Still slow for now.





Prospects for the future

So much work still to be done!

Lots of low hanging fruit and possible questions/projects.

Can we make this more efficient?

- Run faster
- Local learning version
- Better scaling
- Sparse connectivity

Computational advantages of SNNs?

- Fast decision making?
- Robust to noise?
- Robust to adversarial attacks?
- More generalisable?
- Low power (neuromorphic computing)

Answer biological questions?

- What is the role of spikes?
  Just energy efficiency / transmission?
- Local learning rules
- Interaction with synapse/neuron dynamics

Coding it up in PyTorch

Let’s go look at some code.

Crazy idea: massive collaborative project

The surrogate gradient descent tutorial is something that hasn't been done before to my knowledge.

Sound localisation with surrogate gradient descent.

Long history of hand crafted models of sound localisation going back 70 years.

Still not resolved!

So could we turn this into a paper?

Lots of work that we'd need to do.

Lots of separate, somewhat independent things we'd need to check.

So the crazy idea is…

We write a 250 author paper on this.

Everyone is welcome to contribute.

Anyone who contributes a part of a figure or some text is an author (even if this eventually gets removed from the paper).

First/last authors will be based on whoever takes charge of coordinating everything. All other authors in random order.

Ongoing discussions on Discord.

Aim to submit something in 6 months?

Will email all participants a link to sign up.

Join the SNUFA community
https://discord.gg/7y8ed2WgS2

Population Coding in the Cerebellum a Machine Learning Perspective

Reza Shadmehr

April 6th
15:00 CET

Part 2 - Exercise

Fill in the missing parts of the code.

Try to improve the model.

Build a version of the model that uses regression rather than classification.

Understand how the solutions it has found work.